PTS/DeviceNet Interface Software
User's Manual

Issue 3

November 1999

MAN538

Contents

1. Introduction 3
1.1 General 3

2. Configuring the PTS for DeviceNet 4
2.1 Hardware 4
2.2 Software License Key 4
2.3 Configuration Shell 5
2.3.1 Accessing the Shell 5
2.3.2 Changing the Baud Rate 6
2.3.3 Changing the MAC ID 6
2.3.4 Mapping Variables 7
2.3.5 Saving and Restoring the Variable Map 8
2.3.6 Variable Write Behaviour 9
2.3.7 Displaying Statistics 9
2.3.8 Logging Messages 10

3. Programming the PTS 11
3.1 Host I/O 11
3.2 Variables 12
3.3 Arrays 12

4. PTS Device Profile 15
4.1 Overview 15
4.2 Device Description 16
4.3 Object Model 16
4.3.1 Classes 16
4.3.2 Model Description 16
4.4 I/O Access 19
4.4.1 I/O Input Message Format 19
4.4.2 I/O Output Message Format 20
4.5 Single Variable Access 21
4.5.1 Single Variable Read Message Format 21
4.5.2 Single Variable Write Message Format 22
4.6 Block Variable Access 23
4.6.1 Block Variable Read Message Format 23
4.6.2 Block Variable Write Message Format 25

5. Electronic Data Sheet 27

6. Hardware Configuration 29
6.1 DeviceNet Interrupts 29
6.2 DeviceNet Connections 29
6.3 CPU360 Board Layout 30

A Statistics Display 31

B Logged Message Format 33

Copyright © 1999 Quin Systems Limited

1. Introduction

1.1 General

This manual relates to the following versions of software in the PTS unit:

DeviceNet Interface Version 1.3
PTS Host software Version 2.1 or later

This document describes the PTS/DeviceNet interface implemented on the second CANbus interface of the CPU360 (PTS Mk2 or Machine Controller).

DeviceNet is a low level network based on CANbus which is designed to connect industrial devices (such as limit switches, photoelectric sensors, etc.) to a PLC or PC. The PTS DeviceNet implementation allows the PTS to be connected to DeviceNet and appear as a slave device using the Predefined Master/Slave Connection Set of connections. The Predefined Set contains one connection for explicit messages to allow read/write of parameters, variables, etc., and several different I/O connections to allow read/write of discrete I/O bits. Currently the only I/O connection type implemented is a polled connection.

The remote PLC/PC can access a pre-defined set of PTS variables and/or array elements and host level I/O via the I/O connection. The variables/array elements can also be accessed via the explicit message connection. The host level I/O provides 8 groups of virtual I/O bits which can be used for DI lines, SO, CO, RI, RO, etc.

Copyright © 1999 Quin Systems Limited

2. Configuring the PTS for DeviceNet

2.1 Hardware

Before switching the PTS on check that the DeviceNet interrupt link is installed between pins 11 & 12 of jumper J11 as described in section 4.1, DeviceNet Interrupts. Without this jumper link the DeviceNet software will not work.

2.2 Software License Key

The software for the DeviceNet interface will not operate unless a software key has been entered to enable this option. The software key is different for each PTS and can be obtained from your sales office given the system serial number which can be found by using the SK command as shown below.

To enable the software the following command should be entered on Port A (the main programming port) in privileged mode. You enter the text in **bold** while the PTS displays something similar to the rest.

```
1> SK
Serial number: 006545
Feature     Version   Key
New feature ? devicenet
Version ? 1.3
Key ? abcd
OK
```

Note that the feature name (devicenet) must be entered in lower case exactly as shown above. Note also that it is necessary to turn the power off and back on again to run the Devicenet software.

If the software needs to be disabled, first make a note of the software key in case it is needed in the future. Then proceed as above but simply press the Return key in response to the "Version ?" prompt as follows.

```
1> SK
Serial number: 006545
Feature     Version   Key
devicenet     1.1     ABCD
New feature ? devicenet
Version ?
Feature devicenet removed
```

Copyright © 1999 Quin Systems Limited

2.3 Configuration Shell

2.3.1 Accessing the Shell

The DeviceNet configuration shell is a command interface specific to DeviceNet which allows you to configure the MAC ID and baud rate for DeviceNet as well as displaying the communications status and logging DeviceNet message packets for diagnostic purposes.

To enter the shell type the DQ command at Port A (the main programming port) in privileged mode:

```
1> DQ

dns>
```

The dns> prompt shows that the configuration shell is ready for a new command. Typing help makes the shell display a list of available commands as follows:

```
dns> help
baud <baudrate> Set DeviceNet baud rate
clear           Unmap all variables
default         Map default variables
disp <num>      Display <num> logged messages
list            List variable mapping
log <num>       Set message log to <num> messages
log all         Log all messages (default)
log exp         Log explicit messages only
macid <id>      Set DeviceNet MAC Id
map <var> <idx> Map variable (at index)
quit            Quit from this shell
restore         Restore variable map from NVM
save            Save variable map in NVM
stat            Display DeviceNet statistics
unmap <var>     Unmap variable
upload          Output map as commands to logfile
wa <var>        Set variable to write always
woc <var>       Set variable to write on change
help            Display this list
```

If DeviceNet is not enabled an error message is displayed instead:

```
1> DQ

DeviceNet is not enabled

1>
```

In this case you need to enter a software license key as described in the section Software License Key and cycle the power to the PTS to start the DeviceNet software.

Copyright © 1999 Quin Systems Limited

2.3.2 Changing the Baud Rate

The DeviceNet baud rate can be set using the *baud* command. The baud rate can be set to 125, 250 or 500 kB. The following example shows how to change the baud rate from 125 to 500 kB.

```
dns> baud
Baud rate 125k

dns> baud 500
OK
```

Entering the baud command without a value displays the current setting. Entering the command with a value causes the setting to be changed. The shell displays OK when the new value has been validated, communications have been restarted at the new rate and the new setting has been saved to non-volatile memory. If the shell does not display OK then it is likely that the new baud rate is not correct or the PTS is not connected to DeviceNet.

2.3.3 Changing the MAC ID

The MAC ID is the address of a unit on the DeviceNet. Each unit on the network must have a unique MAC ID between 1 and 63. The MAC ID of the PTS can be changed using the *macid* command. The following example shows how to change the MAC ID from 25 to 30.

```
dns> macid
MAC Id 25

dns> macid 30
OK
```

Entering the macid command without a value displays the current setting. Entering the command with a value causes the setting to be changed. The shell displays OK when the new value has been validated, communications have been restarted with the new MACID and the new setting has been saved to non-volatile memory. If the shell does not display OK then it is likely that another unit with the same MAC ID is already on DeviceNet and the duplicate MAC ID check has failed.

Copyright © 1999 Quin Systems Limited

2.3.4 Mapping Variables

You can access up to 50 PTS variables or array elements over DeviceNet. By default these variables are $V1 up to $V50 and are accessed from DeviceNet by their index or instance number. In the default case variable $V1 is instance number 1 and so on up to variable $V50 which is instance number 50. The instance number and name of all the available variables can be displayed by the *list* command as shown in the following example.

```
dns> list
Index Variable  WOC
  1         V1   ON
  2         V2   ON
  3         V3   ON
  4         V4   ON
  5         V5   ON
```

The *default* command can be used to reset the PTS to use the default variables.

```
dns> default
```

If the default set of variables is not what you want it is possible to change which variable or array element is mapped at any index or instance number. For example to map variable $SPD to variable instance number 10 use the *map* command as shown below.

```
dns> map spd 10
dns> list
Index Variable  WOC
  .         ..   ..
 10        SPD   ON
  .         ..   ..
```

In this case accessing variable instance number 10 from DeviceNet actually accesses the PTS variable $SPD. If there is already a variable mapped at instance number 10, the old variable is removed and the new variable replaces it. If the second parameter, the instance number, is omitted from the *map* command the variable is mapped at the first free instance number.

It is also possible to remove a mapped variable or array element using the *unmap* command as shown below.

```
dns> unmap spd[5]
```

This removes array element $SPD[5]. Assuming $SPD[5] was mapped at instance number 10, the *unmap* command leaves instance number 10 without an attached variable. Any writes to instance number 10 will be ignored and any reads will return zero.

The *clear* command is used unmap all variables and array elements. It is normally used to clear the variable map to known empty state before downloading a variable configuration file.

```
dns> clear
```

Copyright © 1999 Quin Systems Limited

2.3.5 Saving and Restoring the Variable Map

Once the variable map has been specified it must be saved to non volatile memory so that it is still available when the system is next switched on. This is done with the *save* command shown below.

```
dns> save
OK
```

If the map is saved successfully the OK message is displayed on the following line.

It is also useful to save the map to a file on a PC. This means that you can configure another machine or a spare in the same way as this one. The *upload* command makes the system output the variable map as a set of *map* commands which can later be downloaded. To use this command, first set your PC to capture the output from the PTS, then type in the *upload* command. The system will output a stream of map commands followed by the dns> prompt as shown below.

```
dns> upload
CLEAR
MAP V1 1
WOC V1
.
.

dns>
```

When the output has finished, save the file to disk. It is a good idea to edit the file to remove the initial upload command and the prompt at the bottom to avoid any errors when the file is down loaded.

The *restore* command restores the variable map from non volatile memory. The map is automatically restored on power up so this command is only needed to restore a known map from non volatile memory when you have been experimenting with different settings. The *restore* command is shown below. If the map is restored successfully the OK message is displayed on the following line.

```
dns> restore
OK
```

Copyright © 1999 Quin Systems Limited

2.3.6 Variable Write Behaviour

When a particular variable is written to from DeviceNet it is often not desirable to write to the mapped PTS variable or array element unless the value has changed. This is because the variable may be a trigger variable and every write by the DeviceNet scanner would cause the trigger variable's command string to be executed. At high scan rates this would put an unnecessary load on the PTS which could slow down more important operations. Because of this variables are set by default to "write on change", in other words the variable is only written to when the value sent by the scanner has changed. This is indicated in the output from the *list* command when the WOC column is set to ON. The alternative is to set the variable to "write always", in other words the variable is always written to when a value is sent by the scanner. In this case the WOC column is set to OFF.

The *woc* command is used to set variables to "write on change" as shown in the following example.

```
dns> woc spd
```

This sets variable $SPD to "write on change". If the variable is omitted from the *woc* command then all current variables and any mapped subsequently are set to "write on change". The *wa* command is used to set variables to "write always" as shown below.

```
dns> wa
Set WOC flag on ALL variables ? (Y/N) y
OK
```

This command sets all current variables and any mapped subsequently to "write always".

2.3.7 Displaying Statistics

The *stat* command allows you to view some statistics of the DeviceNet performance and the state of the connections. The following example shows a typical display:

```
dns> stat
Version 1.1
Packets Received 2  Transmitted 5  Errors 1
Fragment msgs OK 0  NAKS 0  retries 0  timeouts 0
States   Estab Closed Closed
EPRs         0      0      0
```

The display shows the software version followed by total counts of the number of DeviceNet message packets received and transmitted and a count of the number of errors. You can use these numbers to find out how well the DeviceNet link is performing. A full description of the statistics display is given in Appendix 1.

Copyright © 1999 Quin Systems Limited

2.3.8 Logging Messages

For advanced diagnostic work you can log DeviceNet messages to a message buffer in memory. The size of the buffer is limited but it is designed to always hold the last *n* messages where *n* is specified by the *log* command. The following example shows the message buffer being set to hold the last 100 messages:

```
dns> log
Buffer size 0

dns> log 100
```

The messages can be displayed at any time by using the *disp* command. The following example shows the disp command being used to display the last 10 messages:

```
dns> disp 10
2544.187 RX C0:9A C1:5A A0:99 A1:C0 CF:50  02 4C 03 01
01
2544.187 TX C0:A5 C1:65 A0:99 A1:60 CF:48  02 94 0B 02
2551.843 RX C0:9A C1:5A A0:99 A1:C0 CF:60  42 4B 03 01
01 02
2551.843 TX C0:A5 C1:65 A0:99 A1:60 CF:38  42 CB 00
2551.902 RX C0:9A C1:5A A0:99 A1:80 CF:70  02 10 05 01
09 00 00
2551.902 TX C0:A5 C1:65 A0:99 A1:60 CF:48  02 90 00 00
```

In this example only six messages are displayed because only six have arrived since the message buffer was set up. A full description of the message display is given in Appendix 2.

Copyright © 1999 Quin Systems Limited

3. Programming the PTS

3.1 Host I/O

The PTS host I/O bits map onto the DeviceNet discrete I/O bits such that when the master node writes to an output bit the result appears in the corresponding PTS host input bit. Similarly when the master node reads an input bit it gets the contents of the corresponding PTS host output bit. The PTS currently supports 8 host input groups (10 - 17) and 8 host output groups (10 - 17). Note that the host I/O is system wide and is not channel or node specific like the standard I/O.

The host inputs are supported by a subset of the normal input commands as follows:

- BIg:[n] Inhibit function input.
- DIg:n /... Define function input (restricted).
- EIg:[n] Enable function input.
- IIg:n If input true do command line.
- LIg List input line definitions.
- MIg:[n] Mask function input.
- RIg:[n] Read input line(s) in group g.

The host outputs are similarly supported by a subset of the normal output commands as follows:

- COg:[n] Clear output line n in group g.
- IOg:n If output true do command line.
- LOg List output line definitions.
- ROg:[n] Read output line state(s) in group g.
- SOg:[n] Set output line n in group g.

The following example shows host input line 10:5 being defined as a function input to execute sequence 200. When the DeviceNet master sets output number 5 then the corresponding host input is set and the function input is triggered to execute sequence 200.

```
DI10:5+/XS200
```

Copyright © 1999 Quin Systems Limited

3.2 Variables

The variable database is a centralized facility which is accessible to all tasks in the system and holds a set of integer variables. Because variables are generally accessible, it is possible for the user to change a variable via DeviceNet and for the variable to be used subsequently to set a motor parameter in the PTS. Similarly a variable can be set to some motor parameter, such as the position, which can then be read over DeviceNet. A variable can also be set up to trigger execution of a command string on the PTS.

A variable can be set to a constant value using '=' (equals). For example the following command sets the variable $SPD to a value of 5000.

```
1> $SPD=5000
```

A variable can be used in place of a numeric parameter in most commands. For example the following command sets the velocity to the value of the variable $SPD which is currently 5000. If the variable has not been assigned a value, then the "undefined variable" error message is displayed.

```
1> SV$SPD
```

Conversely it is possible to query a parameter and place the result in a variable. The following example updates variable $SPD with the current velocity value.

```
1> $SPD=SV
```

A variable can be defined as a trigger variable so that when it is updated a string of commands is executed. The following example defines $SPD as a trigger variable which causes the velocity to be set to the value of $SPD each time the variable is updated.

```
1> $SPD>CH1/SV$SPD
```

3.3 Arrays

An array is effectively a block of variables which can be referenced by the index number. Before an array can be used it must be created using the IA command, for example:

```
IA$A[10]
```

This creates an array $A with 10 elements $A[1], $A[2], $A[3] up to $A[10]. Each element of the array can be used in the same way as a variable namely as a command parameter, to trigger commands or with query commands. The following example shows a position being read into an array element and then being used in an expression.

```
CH1/$A[1]=DP/RF(100+$A[1])
```

Copyright © 1999 Quin Systems Limited

The array index can be a constant, a variable or an expression. The following example sets the elements of a speed array to ascending values.

```
IA$V[10]
$I=1
$V[$I]=($I*1000)/$I=($I+1)/RP9
```

Arrays are particularly useful with Devicenet when you have a number of different products and you need to set up a number of parameters for each product. Say for example you are designing a cut to length machine which must be able to make 20 different products. For each product there is a cut length, a registration offset and a maximum machine speed. This would normally require 60 variables, a switch sequence and 20 set up sequences similar to the following to change product.

```
ES100
XS(100+$PNO)

ES101
$M01=$L1
RF$R1
SV($S1/2)

ES102
$M01=$L2
....

....

ES120
$M01=$L20
RF$R20
SV($S20/2)
```

Sequence 100 is run when a new product number ($PNO) is selected. This switches to a set up sequence (101 to 120) which sets the master axis length for the Motion Generator, sets the reference offset and sets the initial speed at half the maximum.

If you use arrays, however, the solution becomes much simpler. You only need 3 arrays and a single sequence.

```
IA$LEN[20]
IA$RF[20]
IA$MAX[20]

ES100
$M01=$LEN[$PNO]
RF$RF[$PNO]
SV($MAX[$PNO]/2)
```

You now only need one set up sequence because you can use the product number ($PNO) as an array index to select the correct array element for the product.

Copyright © 1999 Quin Systems Limited

Not only is the solution simpler, it is also much easier to make changes. To increase the number of different products all you have to do is to increase the array sizes and increase the upper limit on $PNO. Sequence 100 does not need to change. Without arrays any change to the number of products means writing a lot of extra sequences.

If you use arrays, changes to the product handling (change the initial speed to 3/4 of maximum for example) involve changing a single sequence. Without arrays you would have to change dozens of sequences and it would be very easy for mistakes to creep in.

Note

In this manual variables are used extensively to map onto Devicenet. In all cases where a variable is called for, an array element can be used instead. It is important to create the arrays using the IA command before they are used. SP saves the array declaration so that it is restored automatically on start up.

Copyright © 1999 Quin Systems Limited

4. PTS Device Profile

4.1 Overview

DeviceNet units are described by a device profile which is a formal definition of the device behaviour, I/O data and configuration data. The device profile consists of the object model (see below), the I/O data format, the configuration data and the interface to that data.

The Object model specifies:

- The components that make up the unit
- The externally visible behaviour of the unit
- How the components fit together to provide the required behaviour
- The information which can be sent to or read from the components

The components mentioned above are represented by Objects in the Object model. Objects which are of the same type are said to belong to the same Class. This means that all the objects in a given class hold the same type of information, provide the same services and implement the same behaviour. From a practical point of view, if you know what attributes an object has and what services it provides, you can use DeviceNet messages to read or write the attributes or invoke the services.

The rest of this chapter is the formal device profile for the PTS.

Copyright © 1999 Quin Systems Limited

4.2 Device Description

The PTS controls the position and velocity of one or more electric motors. In typical PTS applications the position/velocity profiles are either pre-defined or can be calculated at run time so there is no need to control these via DeviceNet. Instead the DeviceNet interface to the PTS operates at a higher level providing access to a set of virtual I/O lines and a set of general purpose variables. The I/O lines can be used for control and signalling while the variables can be used to set operational parameters (such a speed, length of cut) and to return status information.

The PTS is a Group 2 only slave device without UCMM and supports the Poll I/O and Explicit connections from the Predefined Master/Slave Connection Set.

4.3 Object Model

4.3.1 Classes

The table below shows the classes which are supported by the PTS along with the number of objects within each class and a description.

Class	Number of Objects	Description
Assembly	2	The I/O assembly provides access to the Host I/O and variables. The Variable assembly provides access to blocks of variables.
Connection	2	The PTS implements the Poll I/O and Explicit connections from the Predefined Master/Slave Connection Set.
DeviceNet	1	The DeviceNet object provides the configuration and status of the physical connection to DeviceNet.
Identity	1	Provides identification and general information about the PTS.
Message Router	1	Provides a message connection point for all the objects and classes in the PTS.
Variable	0 to 50	Each variable object provides read/write access to a PTS variable. Blocks of variables can be accessed via the variable assembly. Variables can also be accessed via the I/O assembly.

Table 1: DeviceNet Classes

4.3.2 Model Description

The I/O Assembly object provides access to the Host I/O and the variables in the PTS via the Poll I/O connection. The Variable objects provide access to the PTS variables via the Explicit message connection and the Message Router. The Variable Assembly object provides access to blocks of PTS variables.

Copyright © 1999 Quin Systems Limited

The diagram below shows the objects and classes within the PTS/DeviceNet Interface and the connections between them.

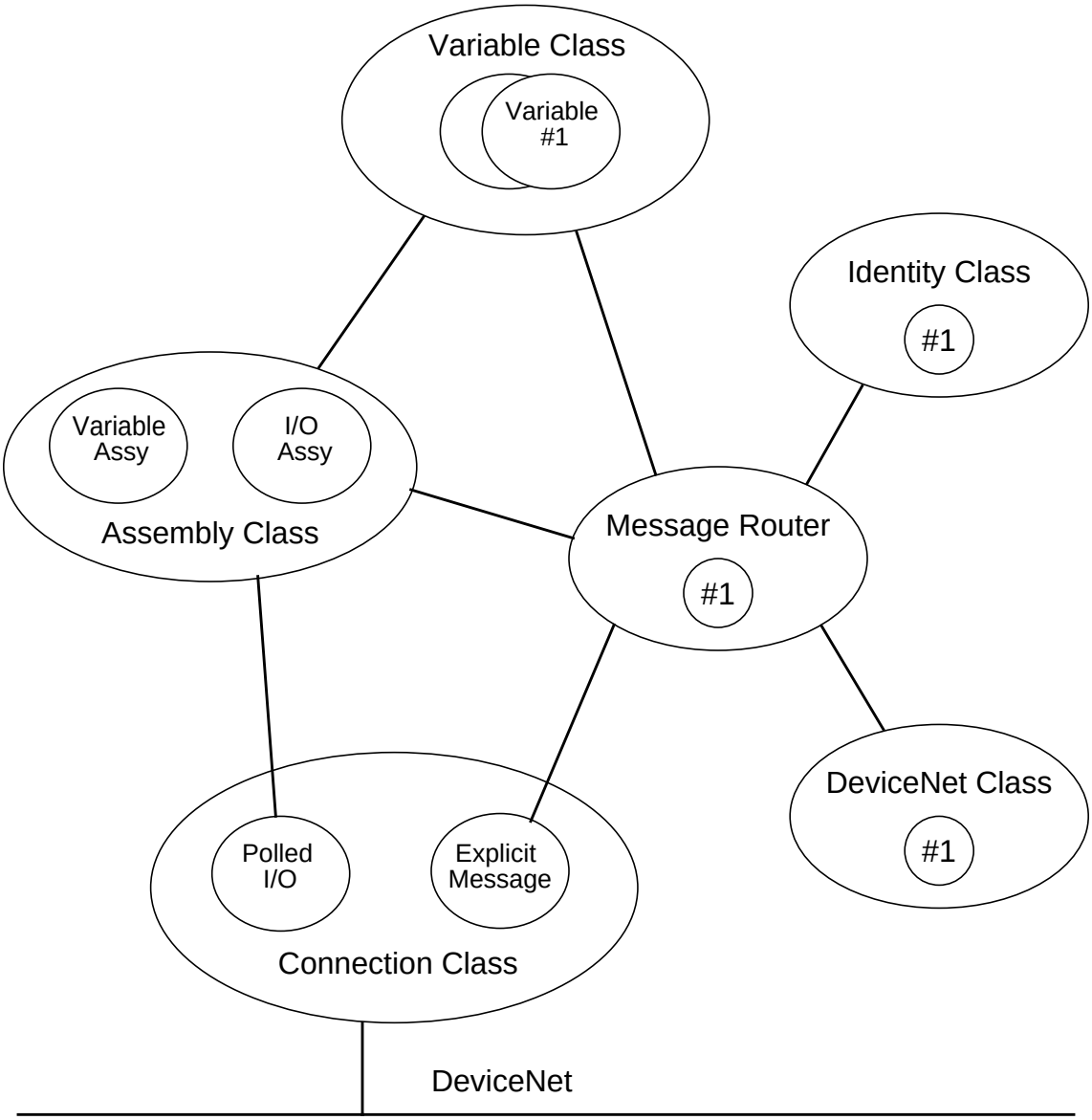


Figure 1. PTS/DeviceNet Interface Object Model

The table below shows the DeviceNet objects along with their class and instance IDs.

Object	Class ID	Instance ID
DeviceNet	03hex	1
Identity	01hex	1
Explicit Connection	05hex	1
I/O Connection	05hex	2

Table 2: PTS Object IDs

Copyright © 1999 Quin Systems Limited

Object	Class ID	Instance ID
Message Router	02hex	1
I/O Assembly	04hex	1
Variable Assembly	04hex	3
Variable	64hex	1 to 50

Table 2: PTS Object IDs

Copyright © 1999 Quin Systems Limited

4.4 I/O Access

4.4.1 I/O Input Message Format

The I/O input command message consists of 8 data bytes which map directly on to the Host I/O virtual input lines. When a bit is set in the command message the corresponding input line is set and vice versa. The message also contains 6 bytes which specify the index or instance number of a variable to be read and the index and value of a variable to be written. The following table shows the correspondence between the bits in the message and the Host input lines and the variables.

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	10:8	10:7	10:6	10:5	10:4	10:3	10:2	10:1
1	11:8	11:7	11:6	11:5	11:4	11:3	11:2	11:1
2	12:8	12:7	12:6	12:5	12:4	12:3	12:2	12:1
3	13:8	13:7	13:6	13:5	13:4	13:3	13:2	13:1
4	14:8	14:7	14:6	14:5	14:4	14:3	14:2	14:1
5	15:8	15:7	15:6	15:5	15:4	15:3	15:2	15:1
6	16:8	16:7	16:6	16:5	16:4	16:3	16:2	16:1
7	17:8	17:7	17:6	17:5	17:4	17:3	17:2	17:1
8	Output variable index (1 to 50)							
9	Input variable index (1 to 50)							
10	Input variable value least significant byte							
11	Input variable value byte 2							
12	Input variable value byte 3							
13	Input variable value most significant byte							

Table 3: I/O Input Message Format

The input variable index specifies the index of a PTS variable or array element to be written and the input variable value bytes specify the value to be written to it. If the index is zero or out of range no variable is written. The output variable index specifies the index of a PTS variable or array element which is to be read and returned in the I/O output response message. If the index is zero or out of range or does not correspond to a valid variable no variable is read.

The mapping between the variable index and the variable name is described in chapter 2, Configuring the PTS for DeviceNet.

Copyright © 1999 Quin Systems Limited

4.4.2 I/O Output Message Format

The I/O output response message consists of 8 data bytes which map directly on to the Host I/O virtual output lines. When an output line is set the corresponding bit in the response message is set and vice versa. The message also contains 6 bytes which hold the index and value of a PTS variable being read by the scanner. The following table shows the correspondence between the bits in the message and the Host output lines and the variable.

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	10:8	10:7	10:6	10:5	10:4	10:3	10:2	10:1
1	11:8	11:7	11:6	11:5	11:4	11:3	11:2	11:1
2	12:8	12:7	12:6	12:5	12:4	12:3	12:2	12:1
3	13:8	13:7	13:6	13:5	13:4	13:3	13:2	13:1
4	14:8	14:7	14:6	14:5	14:4	14:3	14:2	14:1
5	15:8	15:7	15:6	15:5	15:4	15:3	15:2	15:1
6	16:8	16:7	16:6	16:5	16:4	16:3	16:2	16:1
7	17:8	17:7	17:6	17:5	17:4	17:3	17:2	17:1
8	Not used							
9	Output variable index (1 to 50)							
10	Output variable value least significant byte							
11	Output variable value byte 2							
12	Output variable value byte 3							
13	Output variable value most significant byte							

Table 4: I/O Output Message Format

The output variable index corresponds to the output variable or array element index specified in the input command message and indicates which PTS variable is being read. The output variable value bytes give the value of the variable or array element being returned to the scanner. If the output variable index in the output response message is zero then the variable has not been read and the value bytes are not valid. This can occur is no variable was requested (index was zero), the output variable index was out of range, or the requested variable did not exist.

Copyright © 1999 Quin Systems Limited

4.5 Single Variable Access

4.5.1 Single Variable Read Message Format

The single variable read message is sent as an explicit message to the appropriate instance of the Variable class. The instance ID is the same as the variable index shown by the *list* command described in chapter 2, Configuring the PTS for DeviceNet The service code used is READ_VAR (50). The format of the single variable read command message from the master is shown below.

Byte	Contents
0	Message Header
1	R/R = 0, Service Code = 50
2	Class ID = 100
3	Instance ID (1 to 50, see above)

Table 5: Single Variable Read Input Message Format

In response the PTS returns either a message containing the variable value or an error message indicating that the object does not exist. The same error message is also returned if the requested Instance ID is outside the range 1 to 50. The format of the normal response message is shown below.

Byte	Contents
0	Message header
1	R/R = 1, Service code = 50
2	Variable value LS Byte
3	Variable value Byte 2
4	Variable value Byte 1
5	Variable value MS Byte

Table 6: Single Variable Read Output Message Format

The format of the error response is shown below.

Byte	Contents
0	Message header
1	R/R = 1, Service code = 20
2	General error code = 22
3	Additional code (unspecified)

Table 7: Single Variable Read Error Response Format

Copyright © 1999 Quin Systems Limited

4.5.2 Single Variable Write Message Format

The single variable write message is sent as an explicit message to the appropriate instance of the Variable class. The instance ID is the same as the variable index shown by the *list* command described in chapter 2, Configuring the PTS for DeviceNet. The service code used is WRITE_VAR (51). The format of the single variable write command message from the master is shown below.

Byte	Contents
0	Message Header
1	R/R = 0, Service Code = 51
2	Class ID = 100
3	Instance ID (1 to 50, see above)
4	Variable value LS Byte
5	Variable value Byte 2
6	Variable value Byte 1
7	Variable value MS Byte

Table 8: Single Variable Write Input Message Format

In response the PTS returns either a normal response message or an error message. The format of the normal response message is shown below.

Byte	Contents
0	Message header
1	R/R = 1, Service code = 51

Table 9: Single Variable Write Output Message Format

An error message is returned if the requested Instance ID is outside the range 1 to 50. The format of the error response is shown below.

Byte	Contents
0	Message header
1	R/R = 1, Service code = 20
2	General error code = 22
3	Additional code (unspecified)

Table 10: Single Variable Write Error Response Format

Copyright © 1999 Quin Systems Limited

4.6 Block Variable Access

Accessing blocks of variables is done by sending messages to the Variable Assembly object. The read response and the write command messages are usually longer than a single CAN message packet and are therefore fragmented using the acknowledged fragmentation protocol. The read and write command messages specify the block of variables by giving the instance ID of the first variable followed by the number of variables in the block. It is an error to specify either the first variable or the number of variables so that any of the instance IDs is outside the range 1 to 50. This produces an error message indicating an invalid parameter.

4.6.1 Block Variable Read Message Format

The block variable read command message is sent as an explicit message to the Variable Assembly. The command message specifies the instance ID of the first variable and the number of variables to read. The format of the block variable read command is shown below.

Byte	Contents
0	Message header
1	R/R = 0, Service code = 50
2	Class ID = 4
3	Instance ID = 3
4	First variable instance ID (1 to 50)
5	Number of variables

Table 11: Block Variable Read Input Message Format

Copyright © 1999 Quin Systems Limited

If the variable instance IDs are in range and all variables exist, the PTS returns the normal response message which contains the values of each variable.The format of the normal response message is shown below **before** fragmentation.

Byte	Contents
0	Message header
1	RR = 1, Service code = 50
2	Value of 1st variable LS Byte
3	Value of 1st variable Byte 2
4	Value of 1st variable Byte 1
5	Value of 1st variable MS Byte
6	Value of 2nd variable LS Byte
7	Value of 2nd variable Byte 2
8	Value of 2nd variable Byte 1
9	Value of 2nd variable MS Byte
...	More variables

Table 12: Block Variable Read Output Message Format

The errors which may be returned by the PTS instead of the normal response message are shown below.

Code	Meaning
22	Object does not exist. One or more variables in the block is not defined.
32	Invalid parameter. The block has been defined so that one or more variable instance IDs is outside the range 1 to 50.

Table 13: Block Variable Read Error Codes

Copyright © 1999 Quin Systems Limited

4.6.2 Block Variable Write Message Format

The block variable write command message is sent as an explicit message to the Variable Assembly. The command message specifies the instance ID of the first variable and the number of variables to read followed by the values of each variable in the block. The format of the block variable write command is shown below **before** fragmentation.

Byte	Contents
0	Message header
1	R/R = 0, Service code = 51
2	Class ID = 4
3	Instance ID = 3
4	First variable instance ID (1 to 50)
5	Number of variables
6	Value of 1st variable LS Byte
7	Value of 1st variable Byte 2
8	Value of 1st variable Byte 1
9	Value of 1st variable MS Byte
10	Value of 2nd variable LS Byte
11	Value of 2nd variable Byte 2
12	Value of 2nd variable Byte 1
13	Value of 2nd variable MS Byte
...	More variables

Table 14: Block Variable Write Input Message Format

If the variable instance IDs are in range, the PTS returns the normal response message as shown below.

Byte	Contents
0	Message header
1	RR = 1, Service code = 51

Table 15: Block Variable Write Output Message Format

Copyright © 1999 Quin Systems Limited

An error message is returned if any of the variable instance IDs is outside the range 1 to 50. The format of the error response is shown below.

Byte	Contents
0	Message header
1	R/R = 1, Service code = 20
2	General error code = 32
3	Additional code (unspecified)

Table 16: Block Variable Write Error Response Format

Copyright © 1999 Quin Systems Limited

5. Electronic Data Sheet

The information in an Electronic Data Sheet (EDS) allows configuration tools to provide informative screens that guide a user through configuring a DeviceNet device. An up to date copy of the EDS is available from Quin Systems Ltd. The text of the EDS is shown below.

```
$   Quin Systems Ltd.
$   EDS for PTS/DeviceNet interface
$   Revision History
$   1.1 23 Dec 98 Written by John Lambe
$  1.2 02 Mar 99 Modified to include variable access in I/O
connection
$   1.3 09 Nov 99 Returns correct serial number from NVM

[File]
    DescText = "PTS";
    CreateDate = 23-12-98;
    CreateTime = 11:50:00;
    ModDate = 09-11-99;
    ModTime = 14:00:00;
    Revision = 1.3;

[Device]
    VendCode = 455;
    VendName = "Quin Systems Ltd.";
    ProdType = 1;
    ProdTypeStr = "Control Station";
    ProdCode = 1;
    MajRev = 1;
    MinRev = 3;
    ProdName = "PTS";
    Catalog = "";

[IO_Info]
    Default = 0X0001;
    PollInfo = 0X0001, 1, 1;

$ Input Connection
     Input1 =
    14,                         $ 14 bytes produced
    0,                          $ All bits significant
    0x0001,                     $ Poll only
    "Input I/O & Variable",     $ Name string
    6,                          $ Connection path size
   "20 04 24 01 30 03",      $ Assembly Class (4) I/O Instance (1)
                                $ Data Attribute (3)
    "Input I/O & Variable";     $ Help string
```

Copyright © 1999 Quin Systems Limited

```
$ Output Connection
     Output1 =
    14,                         $ 14 bytes consumed
    0,                          $ All bits significant
    0x0001,                     $ Poll only
    "Output I/O & Variable",    $ Name string
    6,                          $ Connection path size
   "20 04 24 01 30 03",     $ Assembly Class (4) I/O Instance (1)
                                $ Data Attribute (3)
    "Output I/O & Variable";    $ Help string

[ParamClass]

[Params]

[EnumPar]

[Groups]
```

Copyright © 1999 Quin Systems Limited

6. Hardware Configuration

6.1 DeviceNet Interrupts

Before attempting to use the PTS/DeviceNet Interface you must ensure that the CANbus interrupt jumper J11 is correctly configured. The link between pins 1 & 2 enables the Servonet port on the lower pair of connectors S6. The link between pins 11 & 12 enables the DeviceNet port on the upper pair of connectors S5. Figure 2 below shows the correct configuration for jumper J11.The location of the jumper pad and the connectors is shown in figure 3.

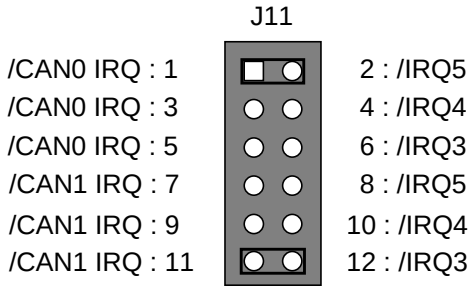


Figure 2. CANbus Interrupt Jumper

6.2 DeviceNet Connections

The connections for the CANbus interface on the front panel 9 way plug and socket S6 are shown below. Note that these comply with the CAN in Automation (CiA) draft standard DS102 Version 2.0, CAN Physical Layer for Industrial Applications.

Pin no.	Signal	Pin no.	Signal
1	Reserved	6	GND
2	CAN_L	7	CAN_H
3	CAN_GND	8	ERROR
4	Reserved	9	CAN_V+ (7–13V)
5	CAN_SHLD (screen)		

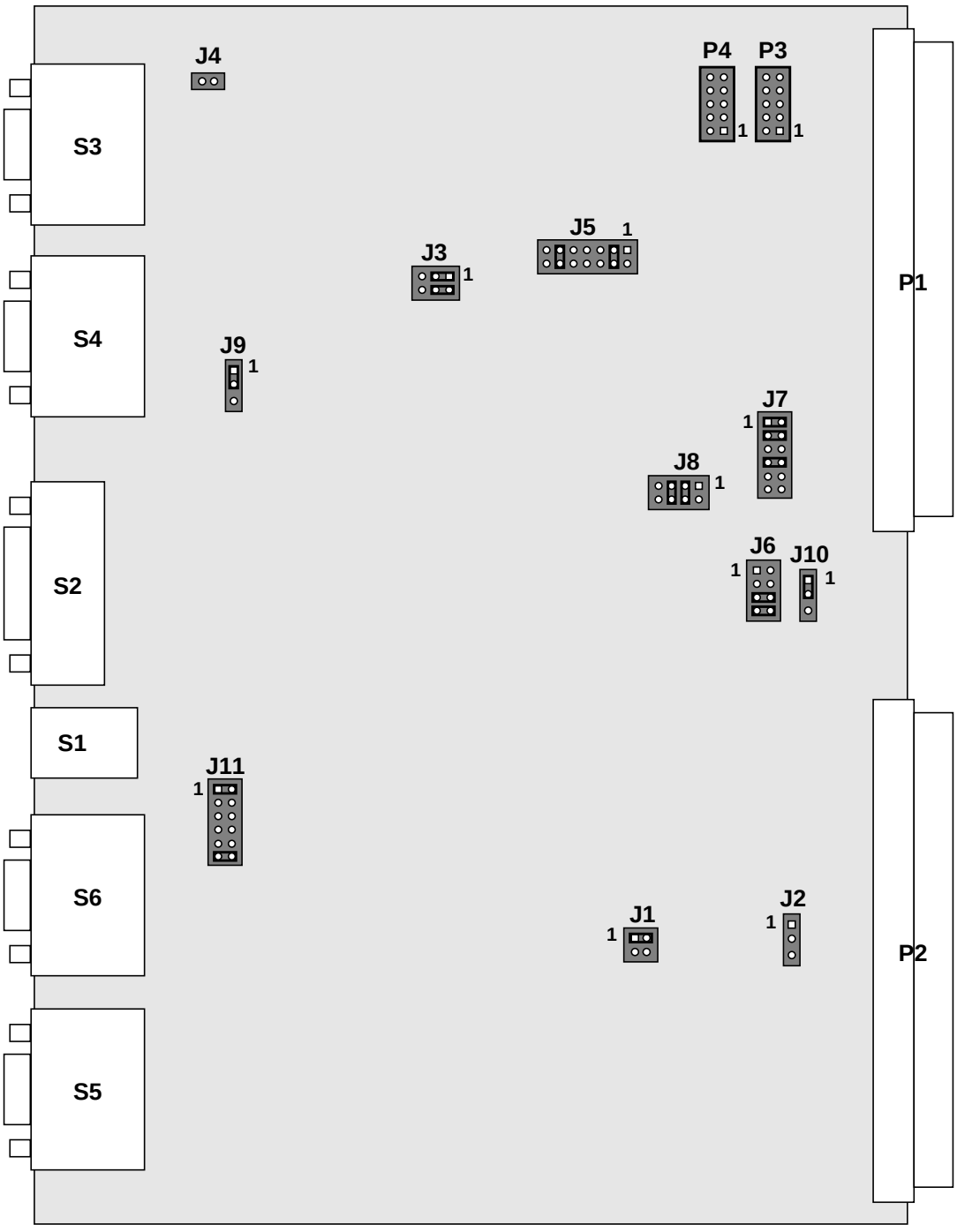
Table 17: DeviceNet Connections

Copyright © 1999 Quin Systems Limited

6.3 CPU360 Board Layout

CPU360 module - component side

Top



Bottom

Figure 3. Jumper and connector locations

Copyright © 1999 Quin Systems Limited

A Statistics Display

The example below shows a typical statistics display produced by the shell command *stat*.

```
      dns> stat
1     Version 1.1
2     Packets Received 2  Transmitted 5  Errors 1
3     Last error was Acknowledgement error
4     Fragment msgs OK 0  NAKS 0  retries 0  timeouts 0
5     States   Estab  Estab Closed
6     EPRs         0   1000      0
```

Line 1 shows the DeviceNet software version, in this case 1.1.

Line 2 shows the number of DeviceNet packets received and transmitted and the number of errors detected since start-up.

Line 3 shows the last error detected. Once the error condition has cleared this line will not be displayed. The error is one of the standard CANbus error conditions as follows:

- Stuff error - More than 5 equal bits have occurred in part of a received message where this is not allowed. Stuff bits help synchronization by adding transitions to the message. A stuff bit is inserted in the bit stream after 5 consecutive equal value bits have been transmitted; the stuff bit being the opposite polarity to the 5 preceding bits.All message fields are stuffed except the CRC, the ACK field and the End of Frame.
- Format error - The fixed format part of a received frame has the wrong format.
- Acknowledgement error - The message transmitted by the PTS was not acknowledged by another node.
- Bit 1/0 error - During transmission of a message (with the exception of the arbitration field) the PTS wanted to send a recessive bit (logic level 1) but the monitored CANbus value was dominant or vice versa.
- CRC error - The CRC received for an incoming message does not match the value calculated by the PTS for the received data.

Line 4 shows the statistics for fragmented messages as follows:

- msgs OK - The number of complete fragmented messages successfully transmitted since start-up. A fragmented message will generally consist of more than 1 message packet.
- NAKS - The number of messages which were NAKed. This means that the receiving node ran out of buffer space for the message.
- retries - The number of transmitted fragments which had to be retried because an ACK was not received within the timeout period.

Copyright © 1999 Quin Systems Limited

- timeouts - The number of fragmented message transmissions which had to be abandoned because an ACK to a retried fragment was not received within the timeout period. If an ACK to a transmitted message fragment is not received within the timeout period, the fragment is retried once. If the retry also times out, the whole message transmission is abandoned.

Line 5 shows the states of the connections. The first column is the Explicit Message connection, the second column is the Polled I/O connection and the third connection is not currently used. Possible states are as follows:

- Closed - The connection does not exist.

- Config - An I/O connection is in the configuring state. This means that the connection has been opened but the expected packet rate (EPR) has not yet been set by the Master node.

- Estab - The connection is established for passing messages. The Explicit Message connection changes to this state once it is opened. The I/O connection changes to this state only after it has been opened and the EPR has been set.

- Tmo - The connection has timed out. An I/O connection will time out if a packet is not received within the time specified by the EPR.

Line 6 shows the expected packet rates (EPR) for the connections in milliseconds. The first column is the EPR for the Explicit Message connection - normally set to zero. The second column is the EPR for the Polled I/O connection - always positive for an established connection. The third connection is not currently used.

Copyright © 1999 Quin Systems Limited

B Logged Message Format

The following example shows a typical logged message display produced by the shell command *disp*:

```
dns> disp 1
2544.187 TX C0:A5 C1:65 A0:99 A1:60 CF:48  02 94 0B 02
```

The fields of the message represented as hexadecimal bytes are as follows:

- Timestamp - The time the message was sent or received in seconds and milliseconds since start-up.
- TX/RX - TX means the message was transmitted by the PTS, RX means it was received.
- C0 - The Control 0 register in the message object structure. Each field in the two Control registers is represented by 2 bits which are read as 01 when the field is zero and 10 when the field is set. The fields in this register are as follows:

7 6	5 4	3 2	1 0
Message Valid	Transmit Interrupt Enable	Receive Interrupt Enable	Interrupt Pending

Table 18:

- C1 - The Control 1 register in the message object structure. The Message Lost field is only valid for a received message. For a transmitted message the field becomes CPU Updating. The fields in this register are as follows:

7 6	5 4	3 2	1 0
Remote Frame Pending	Transmit Request	Message Lost	New Data

Table 19:

- A0/A1 - A0 and the top 3 bits of A1 combine to form the 11 bit CAN message identifier.
- CF - The message configuration register. This contains the data length code (DLC) which is the number of data bytes in the message, the direction code (Dir) which is 1 for transmit and the extended code (Xtd) which is always zero as DeviceNet uses only standard 11-bit identifiers.
- Data - Up to eight bytes of data in hexadecimal format.

Copyright © 1999 Quin Systems Limited

Index

= variable assignment 12

A

acknowledge error 31
arrays 12
  IA command 12
assembly
  I/O 16
  variable 16
assembly class 16

B

baud rate 6
bit 1/0 error 31
block variable
  write 23, 25

C

CANbus interrupt 29
class 15
  assembly 16
  connection 16
  devicenet 16
  identity 16
  message router 16
  variable 16
class diagram 17
class ID 17
classes 16
clear variable map 7
component 15
configuration shell 5
configuring
  hardware 29
  PTS 4
connection
  state 32
connection class 16
CPU360 board layout 30
CRC error 31

D

database 12
default variables 7
device profile 15
devicenet class 16
devicenet connections 29
disp command 33
displaying messages 10
DQ command 5

E

EDS 27
error
  acknowledge 31
  bit 1/0 31
  CRC 31
  format 31
  stuff 31
error messages
  undefined variable 12

F

format error 31
fragmentation 31
fragmented messages 31

H

hardware configuration 29
help command 5
host I/O 11

I

I/O input message 19
I/O output message 20
IA
  create array 12
ID
  class 17
  instance 17
identity class 16
index
  variable 7
input line definitions 11
instance
  variable 7
instance ID 17
interrupt jumper 4, 29
introduction 3

J

J11 29
jumper

Copyright © 1999 Quin Systems Limited

interrupt 4, 29

L

LI 11
license key 4
disable 4
enable 4
list
input line definitions 11
list variables 7
logged message format 33
logging messages 10

M

MAC ID 6
map variables 7
mapping
variable 7
message
block variable write 23, 25
I/O input 19
I/O output 20
single variable read 21
single variable write 22
message router class 16

O

object model 15, 16

P

programming
PTS 11

R

restore
variable map 8
restore variable map 8

S

save
variable map 8
save variable map 8
shell 5
accessing 5
baud 6
clear 7
default 7
disp 10, 33
help 5
list 7
log 10
macid 6
map 7
restore 8
save 8
stat 9, 31
unmap 7
upload 8
wa 9
woc 9
single variable
read 21
write 22
SK command 4
software license key 4
software versions 3
stat command 31
statistics 9
statistics display 31
stuff error 31

T

trigger variable 12

U

undefined variable 12
unmap variable 7
upload
variable map 8

V

variable
default 7
index 7
instance 7
list 7
map 7
mapping 7
unmap 7
write always 9
write behaviour 9
variable class 16
variable map
clear 7
restore 8
save 8
upload 8
variables
as parameters 12
assignment 12
query command 12
trigger 12

Copyright © 1999 Quin Systems Limited

W

write always 9
write on change 9

Copyright © 1999 Quin Systems Limited